COBB

INSIDE DOS

Tips & techniques for MS-DOS & PC-DOS Versions 5 & 6

January 1994
Vol. 5 No. 1

A new version of DOS? Already?

VERSION 6.2

By Van Wolverton

As we get older, time passes more quickly—at least it seems to—but didn't we just get DOS 6.0? Can it really be time for a new version already? Six months is hardly a decent interval; 18 months is more typical, and sometimes even that seems too short. Why the rush?

It all goes back to the problem we discussed in the August 1993 article "DOS 6 Won't Lose Your Data": stories of lost files and trashed disks caused by DoubleSpace, the file-compression utility introduced in Version 6.0. Even though the incidents were rare, difficult (if not impossible) to reproduce, and undoubtedly exaggerated, Microsoft nonetheless found itself with a public relations crisis on its hands. Microsoft had to do something to quell the stories and restore consumer confidence.

Consequently, Microsoft released a new version, 6.2, whose most significant changes deal with DoubleSpace. One of those changes, however, is a disk diagnostic and repair program that goes far beyond the limited capability of CHKDSK and is useful whether or not you use DoubleSpace. Other changes are sprinkled throughout DOS and aren't particularly earthshaking but are helpful.

Safety features for DoubleSpace

Although there were no obvious errors in the way DoubleSpace went about its business, Microsoft took some steps to add more safety features to file compression. In addition to the disk diagnostic and repair program, called ScanDisk, Microsoft added a memory-checking feature to DoubleSpace that makes sure the data to be written to a compressed disk is correct.

ScanDisk is what we wish CHKDSK had been from the beginning. It checks the file structure and can actually repair many of the problems (such as cross-linked files) that CHKDSK can only detect. ScanDisk also checks the surface of the disk. If it detects an area where the strength of the recorded data isn't up to standards—indicating the possibility of surface damage or a flaw in the coating—it moves the data to an area of the disk it has already confirmed is safe and marks the bad area so DOS won't use it.

The file structure scan takes a bit longer than CHKDSK to run, and the surface scan takes about 10 minutes for each 100 megabytes of disk capacity (uncompressed). Unless your system is in use 60 hours a week or more, a monthly surface scan will probably give you sufficient warning of possible problems. Because the file structure scan doesn't take very long, it's a good idea to run it every day, especially if you're using DoubleSpace.

Microsoft added ScanDisk because it found that some uncommon hard disk problems—especially ones that result from damaged file structures or marginal data recording—can cause DoubleSpace problems. ScanDisk improves the reliability of DoubleSpace in several ways. First, it automatically checks the file structure and disk surface the first time you run DoubleSpace to compress a disk. If ScanDisk finds problems, it tells you to run ScanDisk from the DOS prompt to correct the problems, then it exits DoubleSpace; you can't compress the disk until you correct the problems. After you've compressed a disk, you can use ScanDisk to check the file structure of the compressed volume, too, letting you correct problems before they might affect DoubleSpace.

IN THIS ISSUE

DoubleSpace also gains some convenience features with the new version. Its memory requirements are reduced from 43Kb to 37Kb, and it automatically mounts compressed volumes (unless you disable this feature to save another 4Kb of memory). Also, you can uninstall DoubleSpace with one command. If you were concerned about trusting your files to DoubleSpace, these features should offer reassurance that your data is safe.

A safety feature for the MOVE, COPY, and XCOPY commands

One of the most irritating and dangerous gotchas in DOS is the trap lurking in the MOVE, COPY, and XCOPY commands that lets you irretrievably wipe out a file if the target file you specify already exists. DOS replaces the target file with the source file with no warning or acknowledgement.

Microsoft has removed this trap in 6.2. Now, if you try to move or copy one file on top of another, DOS prompts you for confirmation before completing the command.

There's still a loophole, however: Version 6.2 issues the confirmation prompt only if you enter the MOVE, COPY, or XCOPY command at the command prompt; if the command is in a batch file or in a DOSKEY macro, it works as in earlier versions. This means that you don't have to rewrite any batch files that use these commands, but it also means you can still lose files.

More readable numbers

A seemingly small change significantly improves the readability of some screens. The numbers in the output of the DIR, CHKDSK, MEM, and FORMAT commands showing the amount of disk space used and available now include thousands separators (commas in North America). With the ever-increasing size of memory and hard disks, the numbers were getting pretty hard to decipher. Now, for example, you'll see results like this:

```
 Volume in drive D is RUBICON
 Volume Serial Number is 1B59-8336
 Directory of C:\PM5

TE         EXE          22,016     04-16-93   2:26p
PM5        EXE          25,088     06-02-93  10:15a
TEAPP      EXE         274,896     05-27-93  11:02a
PM5APP     EXE       2,767,296     06-12-93   4:20p
    4 file(s) 3,089,296 bytes
              20,090,880 bytes free
```


INSIDE DOS

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group.

Prices: Domestic: $49/yr ($6.00 each); Outside US: $69/yr ($8.50 each)

Phone: Toll free: (800) 223-8720; Local: (502) 491-1900; Customer Relations Fax: (502) 491-8050; Editorial Department Fax: (502) 491-4200

Address: You may address tips, special requests, and other correspondence to

The Editor, *Inside DOS*
9420 Bunsen Parkway, Suite 300
Louisville, Kentucky 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, Kentucky 40220

Copyright: Copyright © 1994, The Cobb Group. All rights reserved. *Inside DOS* is an independently produced publication of The Cobb Group. The Cobb Group reserves the right, with respect to submissions, to revise, republish, and authorize its readers to use the information submitted for both personal and commercial use.

The Cobb Group, its logo, and Satisfaction Guaranteed statement and seal are registered trademarks of The Cobb Group. *Inside DOS* is a trademark of The Cobb Group. Microsoft and MS-DOS are registered trademarks and Microsoft Windows is a trademark of Microsoft Corporation. PC-DOS is a trademark of IBM Corporation.

Postmaster: Second class postage is pending in Louisville, KY. Send address changes to

Inside DOS
P.O. Box 35160
Louisville, KY 40232

Authorized Canada Post International Publications Mail (Canadian Distribution) Sales Agreement #XXXXXX CANADA GST #123669673. Send returns to Canadian Direct Mailing Sys. Ltd., 920 Mercer Street, Windsor, Ontario, N9A 7C2. Printed in the USA.

Staff:

Editor-in-Chief:	Janice Walter
Contributing Editors:	Charity Edelen
	Van Wolverton
Editing:	Meredith Little
	Linda Recktenwald
	Tiffany Taylor
Production Artists:	Tre Pryor
	Laurie A. Wolberton
Design:	Karl Feige
Publications Manager:	Maureen Spencer
Managing Editor:	Mike Stephens
Circulation Manager:	Brent Shean
Publications Director:	Linda Baughman
Editorial Director:	Jeff Yocom
Publishers:	Mark Crane
	Jon Pyles

Advertising: For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Back Issues: To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $6.00 each, $8.50 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you.

Advisory Board: Earl Berry Jr., Tina Covington, Marvin D. Livingood

The CHKDSK output benefits even more:

```
Volume RUBICON  created 10-08-1993 1:31p
Volume Serial Number is 1B59-83F9

      159,268,864  bytes total disk space
       24,231,936  bytes in 7 hidden files
          270,336  bytes in 61 directories
       64,872,448  bytes in 1,659 user files
       69,894,144  bytes available on disk

            4,096  bytes in each allocation unit
           38,884  total allocation units on disk
           17,064  available allocation units on disk

          655,360  total bytes memory
          633,952  bytes free
```

DISKCOPY, startup options, and DEFRAG

Other minor changes tidy up some things Microsoft should have taken care of a few releases ago. For example, you no longer have to swap source and target diskettes when you're doing a one-disk copy. DISKCOPY uses space on the hard disk to copy the source diskette in one gulp, then writes it to the target diskette.

The interactive startup options that first appeared in Version 6.0 are expanded a bit in 6.2. When you press [F8] to step through CONFIG.SYS command by command, you also step through AUTOEXEC.BAT and any batch files it calls. This can be a convenience or a pain, depending on how badly you want to step through these batch files. There are ways to short-circuit the process, however: If you want to skip the rest of the commands at any point, press [F5]; if you want to execute the rest of the commands without being prompted, press [Esc]. You can also start your system without running DoubleSpace by pressing [Ctrl][F5].

Finally, DEFRAG can handle disks with more files (about 20,000) than it could before. Also, Version 6.2 allows SmartDrive to cache CD-ROMs.

The Shell is gone

Not all the changes are additions. Microsoft has removed the DOS Shell, introduced in Version 5.0, from the distribution disks. To get the DOS Shell, other programs such as Edlin, the sample BASIC programs, or commands such as COMP and RECOVER that have fallen from Microsoft's favor, you'll have to order the supplemental diskette from Microsoft or download the files from CompuServe or Microsoft's Download Service—(206) 936-6735.

Of course, most users won't bother, so for all practical purposes, the Shell is no longer part of DOS. If you want to retain the Shell, copy its files (DOSSHELL.*) into a directory other than \DOS. After you upgrade, copy DOSSHELL.* back into DOS and you'll still be able to use the Shell.

Should you upgrade?

If you upgrade to Version 6.2 from 6.0, you'll get a one-disk *stepup* package that simply modifies the 6.0 program files (that's why the stepup won't work with any other version—it needs those 6.0 program files). This upgrade package costs only $9.95 and includes no additional documentation—you'll have to make do with the 6.0 User's Guide.

If you upgrade to 6.2 from Version 5.0 or earlier, you'll get a complete set of installation disks and the User's Guide. You'll pay the same price you would have paid for Version 6.0: $129 manufacturer's suggested retail price, discounted to a $50 to $60 street price.

If you've already upgraded to 6.0, you should upgrade to 6.2. You can buy it for less than $10, or you can download it for free from Microsoft's Download Service at (206) 936-6735 or from CompuServe. ScanDisk alone is worth the ten bucks; the defanged MOVE, COPY, and XCOPY commands and the commas in large numbers are gravy.

If you're still using DOS 5.0 because you prefer more complete third-party utilities such as Stacker, PC Tools Deluxe, or Norton Utilities, DOS 6.2 doesn't add any compelling reasons to upgrade. If you're happy without 6.0, you'll be happy without 6.2.

Contributing editor Van Wolverton has worked for IBM and Intel and is the author of the books Running MS-DOS 5 *and* Supercharging MS-DOS.

Microsoft Diagnostics (MSD) not available in DOS 5

In our November 1993 article "Who Knows Things About Your System That You Don't? MSD, That's Who," we inadvertently noted that Microsoft Diagnostics (MSD) is available in both DOS 5 and 6. Actually, it comes with DOS 6.0, Windows 3.1, and various Windows-based software, but it wasn't included with DOS 5.

You can download MSD Version 2.0 from Microsoft's Download Service at (206) 936-6735 (self-extracting file GA0363.EXE) or through CompuServe's Microsoft Library (zipped file GA0363.ZIP). Or, you can get a free copy (General Application Note GA0363) from Microsoft Product Support Services at (206) 454-2030. We're sorry for any inconvenience this may have caused.

VERSIONS 5 & 6.x

Understanding the basics of your PC's memory

Most DOS users seldom think about the working memory (random access memory, or RAM) in their computer until they upgrade their system or install new software. When you consider buying a more powerful machine, for example, you might notice you have to pay several hundred dollars more for a few extra megabytes of RAM. You might wonder if the additional RAM is worth the expense.

The issue of RAM often comes up in a more jarring way when you try to run new software, only to see a message like *Not enough memory to run application*. When you're faced with the prospect of increasing RAM—by choice or out of necessity—you might become confused by the different components of RAM and the acronym-ridden terminology used to discuss memory. In this article, we'll show you why RAM is important, explain what components make up RAM, and answer other common questions about the memory in your system. Let's begin by looking at some basic RAM characteristics.

What is RAM?

As we mentioned, RAM stands for *random access memory*, but this might not mean much more to you than the acronym. Let's look more closely at some important characteristics of RAM. The RAM in your computer

- allows your system to read and write information
- allows programs to access information randomly
- holds the instructions of programs you run
- lets you make changes without writing to the disk
- loses information when you turn off your PC.

Reading and writing

Perhaps the most basic aspect of RAM is that DOS and other programs can both read information from and write information to RAM. As you might know, this differs from read-only memory (ROM), which stores the basic information your system needs in order to run. ROM is perfect for storing the fundamental instructions your computer needs for booting up, but as we'll see, RAM becomes more important when you run applications.

Random access

As you might guess, *random access* is another important characteristic of RAM. It allows DOS to get information from this type of memory randomly. In other words, DOS doesn't have to read each character in RAM (a process called *sequential access*) to find the information it's looking for. Random access lets DOS read information from RAM very quickly.

For example, imagine if you had to locate the name *Clyde Zellers* in a phone book using sequential access. You'd have to begin with the first entry under *A* and scan hundreds of thousands of entries until you reached *Zellers*. But intuitively, you'd know to use random access to locate this name: You'd immediately turn to the back of the phone book to find the *Z* listings; then you'd refine your search to entries beginning with *Ze* until you finally located *Zellers, Clyde*.

RAM locates information in a similarly efficient manner. However, instead of using the alphabet to organize its information, RAM organizes programs and other files by their address, or location in memory.

Running programs

While RAM rarely performs tasks as simple as looking up a phone number, the analogy gives you an idea of how efficiently RAM works. In practice, one of the most important tasks RAM performs is holding the instructions for your programs. For example, when you start Microsoft Excel, DOS loads into RAM the instructions that make up the program EXCEL.EXE. Then, when you issue commands in Excel, the program doesn't have to go back to the disk to look for instructions because EXCEL.EXE is already in RAM.

Making changes

Although running programs is RAM's most important job, it also performs some secondary functions. One of these secondary functions is serving as a temporary place for you to store and manipulate data. For example, when you're editing a document in a word processing program, the program will load all or part of your document into RAM. If it didn't, you'd notice delays as you entered changes because your computer would have to access the hard disk as you typed each character.

Temporary storage

When you edit a file, you're really making changes to a copy of the file in RAM. You'll need to issue the program's Save command to save your changes to the disk copy of the file. This brings us to RAM's Achilles' heel: Although RAM lets you run programs and quickly access and process data, it isn't a permanent storage area. Unlike your system's hard disk, RAM "forgets" everything when you turn off your PC or

reboot. That's why it's important to save your work frequently when you're entering or changing information. By saving a file, you write the new version to the hard disk, ensuring that your changes permanently become part of the file.

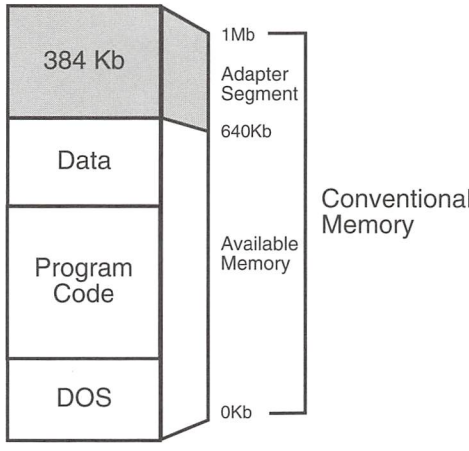
What components make up RAM?

RAM usually consists of conventional memory and either extended memory, expanded memory, or both. Let's explore each component in detail.

Conventional memory

Technically, conventional memory is the first megabyte (1,024Kb) of RAM installed on all personal computers. As you can see in Figure A, however, the last 384Kb of this first megabyte make up the adapter segment—an area of memory usually reserved for hardware and system device drivers. This leaves the first 640Kb for DOS and applications that run under DOS to use. Thus, the term *conventional memory* often refers only to the first 640Kb of RAM.

Figure A



The first megabyte of computer memory is divided into the adapter segment and the 640Kb of memory available to DOS and applications.

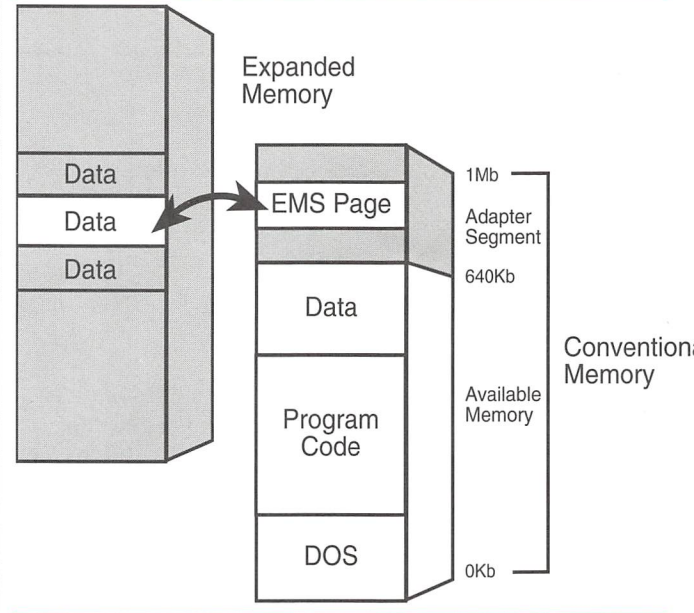
In the early days of personal computers, 640Kb of memory was more than adequate to run most software. However, as software grew more complex, their powerful new features desperately needed more than 640Kb of memory. To address this need, expanded memory came into being.

Expanded memory (EMS)

To allow a computer using an 8088 processor to access more memory than its 640Kb of conventional memory, Lotus, Intel, and Microsoft jointly developed the Lotus-Intel-Microsoft Expanded Memory Specification (LIM EMS). LIM EMS uses an expanded memory add-in board in conjunction with an expanded memory manager to provide your system with additional memory. The expanded memory manager is a file you install on your boot disk that manages the way data moves into and out of the memory on the expanded memory board.

The memory on the expanded memory board is divided into segments, or *pages*, of 16Kb each. Once you've installed the expanded memory board and the expanded memory manager, DOS can use the expanded memory by making requests to the expanded memory manager. As Figure B illustrates, the expanded memory manager creates a "window" in an unused section of the adapter segment in conventional memory. It uses this window to transfer pages of data between expanded memory and conventional memory.

Figure B

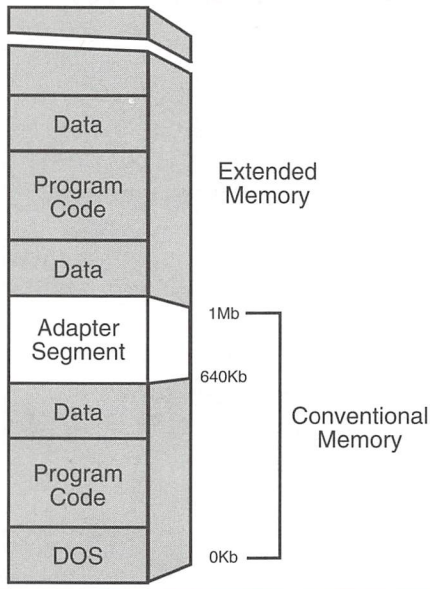


The expanded memory manager swaps pages of information from expanded memory into and out of conventional memory.

Extended memory (XMS)

In addition to expanded memory, computers based on the 286, 386, and 486 processors can use extended memory. Extended memory is memory stacked above the first megabyte of conventional memory, as shown in Figure C on the next page. It's a logical extension of the processor's total address space. Most systems now come with at least some extended memory already installed, but you aren't limited to what comes with your system. Depending on the model of your computer, you can add more extended memory by plugging memory chips into empty slots on the motherboard (the circuit board that distributes power to your computer's chips) or by installing a memory board.

Figure C



Extended memory is the memory that lies above the 1Mb line in a 286-, 386- or 486-based computer.

Although DOS can't access extended memory directly, it can use the HIMEM.SYS driver to access it. You install this driver by placing the following line in your CONFIG.SYS file:

```
device=c:\dos\himem.sys
```

In general, we recommend that you place this line at the top of your CONFIG.SYS file. After loading the HIMEM.SYS driver, your computer will be able to use extended memory for other CONFIG.SYS statements, if necessary. (Note that since you can't load anything into upper memory until after you load HIMEM.SYS, you can't use the DEVICEHIGH directive to load this driver.) After adding this line and saving the CONFIG.SYS file, you need to reboot.

How much RAM do you have?

If you don't know how much memory is installed in your system, you can find out by issuing the MEM command. If you use DOS 5, this command displays a report similar to the one shown in Figure D. If you use DOS 6, MEM displays a report similar to the one shown in Figure E.

DOS 6's memory report indicates the amount of RAM you have on the *Total memory* line, in the *Total* column. To convert this number to megabytes (the unit usually used in reference to RAM), divide it by 1,024. The report in Figure E indicates 4,096Kb total memory; when we divide this number by 1,024, we get 4Mb RAM.

Figure D

```
C:\>mem

   655360 bytes total conventional memory
   654336 bytes available to MS-DOS
   553712 largest executable program size

  3538944 bytes total contiguous extended memory
  3538944 bytes available contiguous extended memory
```

The MEM command in DOS 5 displays this type of memory report.

Figure E

```
C:\>mem

Memory Type          Total  =     Used  +     Free
----------------     ------      ------      ------
Conventional           640K        168K        472K
Upper                    0K          0K          0K
Adapter RAM/ROM          0K          0K          0K
Extended (XMS)        3456K       1088K       2368K
----------------     ------      ------      ------
Total memory          4096K       1256K       2840K

Total under 1MB        640K        168K        472K

Largest executable program size     472K (483248 bytes)
Largest free upper memory block       0K      (0 bytes)
The high memory area is available.
```

The MEM command in DOS 6 displays this type of memory report.

DOS 5's memory report is less straightforward. To figure out how much RAM you have from this report, add the number on the *bytes total conventional memory* line to the number on the *bytes total contiguous extended memory* line. This will give you the amount of RAM in bytes. To convert this figure to megabytes, divide it by 1,024 twice (or by 1,048,576 once). The report in Figure D indicates

```
   655360 bytes total conventional memory
```

and

```
  3538944 bytes total contiguous extended memory
```

for a total of 4,194,304 bytes. Converting this total to megabytes, we get 4Mb RAM.

VERSIONS 5 & 6.x

Manipulating memory to your advantage

In "Understanding the Basics of Your PC's Memory," on page 4, we provided the information you need to have a basic understanding of the role memory plays in your computer. Now that you know the basics, you're ready to make the most of the memory you have in your system. You can use third-party utilities to enhance your system's memory, but before you do, make sure you're taking full advantage of the memory management tools DOS provides. In this article, we'll show you some ways to use the tools that come with DOS to manipulate your PC's memory.

Loading DOS into the high memory area

If your PC uses a 286, 386, or 486 processor and is equipped with at least 64Kb of extended memory, you can clear up to 64Kb more conventional memory to run applications by loading DOS into the high memory area (HMA). This is the area of extended memory that lies just above the first megabyte of RAM. To load DOS into the HMA, add the directive

```
DOS=HIGH
```

after the directive that loads HIMEM.SYS. When you do, the size of the largest program DOS can execute will increase by up to 64Kb.

Another advantage of using this directive comes when you use the BUFFERS directive to set up a disk cache. Normally, this directive reserves 532 bytes of conventional memory for *each* buffer. However, when you load DOS into the HMA, the BUFFERS directive reserves space in the HMA as well, requiring only 512 bytes *total* of conventional memory. (After DOS loads into the HMA, there's room for about 48 buffers.)

Managing memory with EMM386.EXE

EMM386.EXE—the built-in memory manager that comes with DOS Versions 5 and 6—serves two main functions:

- EMM386.EXE can render unused portions of the adapter segment (the area of memory that's located between 640Kb and 1Mb) usable for memory-resident programs and for device drivers.
- EMM386.EXE can emulate expanded memory (EMS) in extended memory (XMS).

Keep in mind that in order to manage memory with EMM386.EXE, your computer must use a 386 or 486 processor and be equipped with extended memory. DOS 6's MemMaker utility automatically installs EMM386.EXE, activating either or both of these functions. If you run MemMaker, you won't need to add the lines to your CONFIG.SYS file that we recommend in the following sections. For more information on using MemMaker, see "MEMMAKER Can Free Up Conventional Memory," in the June 1993 issue.

Taking advantage of unused portions of the adapter segment

As we mentioned in "Understanding the Basics of Your PC's Memory," hardware and system device drivers are usually the only elements that can occupy memory in the adapter segment (also called the *upper memory area*). By installing EMM386.EXE, however, you can enable DOS to use unoccupied portions of this memory area to load terminate-and-stay-resident programs (TSRs) as well as other device drivers, thereby freeing up more conventional memory to run applications. On a system with at least 384Kb of extended memory, EMM386.EXE can reclaim between 30Kb and 130Kb of RAM in this area. To enable this feature, add one of the following lines to your CONFIG.SYS file:

```
device=c:\dos\emm386.exe noems
```

or

```
device=c:\dos\emm386.exe ram
```

Since EMM386.EXE backfills the upper memory area with RAM taken from extended memory, make sure you place its DEVICE directive after the directive that loads HIMEM.SYS. (Recall that DOS must load HIMEM.SYS before it can access extended memory.)

The first option enables DOS to create the upper memory blocks (UMBs) it needs to load TSRs and device drivers while disabling the other main function of EMM386.EXE—emulating expanded memory in extended memory. The second option provides both UMB support and expanded memory emulation.

Once CONFIG.SYS loads HIMEM.SYS and EMM386.EXE to provide UMB support, you need one more directive to enable DOS to use the UMBs EMM386.EXE creates:

```
DOS=UMB
```

If you want to load DOS into the HMA *and* enable it to use UMBs, use the following directive instead:

```
DOS=HIGH,UMB
```

Now you can load TSRs and device drivers into upper memory rather than conventional memory. The DEVICEHIGH directive (used in the CONFIG.SYS file) loads a device driver into upper memory. The LOADHIGH command (used at the command line, in your AUTOEXEC.BAT file, or in any batch file) loads a TSR into upper memory. (If you run MemMaker, it will take care of loading TSRs and device drivers into upper memory so you won't have to load them individually.)

Using extended memory as expanded memory

Although newer programs more commonly use extended memory instead of expanded memory, some older applications *require* expanded memory. You can run such an application—even if all you have is extended memory—if you install EMM386.EXE to emulate expanded memory in extended memory. To do so, load EMM386.EXE with the directive

```
device=c:\dos\emm386.exe
```

or

```
device=c:\dos\emm386.exe ram
```

Either line reserves a default of 256Kb of extended memory to act as expanded memory and causes EMM386.EXE to function as an expanded memory manager. However, instead of storing data in expanded memory, EMM386.EXE simply stores it in extended memory. The first option provides expanded memory emulation without creating UMBs. The second option, as we mentioned above, enables both functions of EMM386.EXE.

You can change the amount of emulated expanded memory by adding after *emm386.exe* in either line a number from 16 to 32,768 (or the upper limit of your extended memory) that represents the number of kilobytes of expanded memory you want. You can also limit the amount of emulated expanded memory by adding to the end of this directive *L=* and a number representing the minimum amount of extended memory (in kilobytes) you want to remain after EMM386.EXE loads. If the sum of the emulated expanded memory and the minimum extended memory you specify exceeds the total amount of extended memory in your system, EMM386.EXE will emulate only as much expanded memory as the minimum extended memory will allow.

A minor conflict to watch out for

If you decide to use EMM386.EXE to emulate expanded memory in extended memory, it will consume 64Kb of space in the adapter segment to create the window expanded memory uses to transfer data into and out of conventional memory. This reduces the amount of unused space in upper memory, thereby reducing the number of UMBs EMM386.EXE can create and ultimately the number of device drivers and TSRs you can load with DEVICEHIGH and LOADHIGH. On a system with 130Kb of reclaimable upper memory, this probably won't pose a problem. On a system with less than 64Kb of reclaimable upper memory, you'll have to load some of the TSRs and device drivers into conventional memory (using DEVICE and LOAD instead of DEVICEHIGH and LOADHIGH) if you want to emulate expanded memory.

To check how much upper memory is left after loading the TSRs and device drivers but before emulating expanded memory, issue the command

```
C:\>mem /c
```

Then, look for the line that reports the total free upper memory. In DOS 5's report, this is the line under *Upper Memory :* that says *Total FREE :*. In DOS 6's report, this is the line in the Memory Summary table that begins with *Upper*. The figure for free upper memory appears in the last column of this table.

Conclusion

If you experience problems with having too little memory or not having the kind you need to run certain applications, the suggestions we presented in this article might help. Specifically, we showed how you can

- load DOS into the high memory area to free up conventional memory
- load EMM386.EXE to take advantage of the unused space in the upper memory area, and then load device drivers and

TSRs into this area instead of into conventional memory

- load EMM386.EXE to emulate expanded memory in extended memory when you need to run a program in expanded memory but you don't have any.

The difference in how many and what programs you can run may surprise you—and save you money.

BATCH FILE TECHNIQUE

VERSIONS 5 & 6.x

Using NUKE.BAT to permanently delete files

In the days before DOS 5, you had to use a third-party utility program to recover a deleted file. Since DOS 5, you can use the UNDELETE command to recover a file if you act immediately. However, there are times when you want to delete a file or a group of files permanently. Even if it's unlikely that someone will try to read your deleted files, it would still be nice to know that the files you've deleted will remain deleted—and unrecoverable.

Although you can use DOS 6's DELTREE command to permanently remove a directory and its contents, there's no easy way to eliminate individual files in either DOS 5 or 6. In this article, we'll show you a batch file, NUKE.BAT, that will permanently delete the file or group of files you specify.

The NUKE batch file

NUKE.BAT is a simple batch file you can use to delete individual files or groups of files from your hard disk or from a floppy disk. You can execute the batch file from any drive or directory as long as you include the complete path to the file or files you want to delete.

NUKE.BAT creates a zero-byte file and then copies it over the files you specify. The beauty of this technique is that when you copy a zero-byte file to another file, it destroys the target file's contents and then deletes the file.

First, we'll create NUKE.BAT and show you how it works. Then we'll show you how to use NUKE.BAT and run through a few examples. Finally, we'll double-check the files we've removed by attempting to undelete them. Of course, you should use this batch file carefully, because once you remove the files, they're gone. UNDELETE won't bring back the information stored in them. Now, let's create NUKE.BAT.

Building NUKE.BAT

You can use the DOS Editor or another ASCII-compatible word processor to create NUKE.BAT. To create the file, type in the code shown in Figure A. Then, save it under the name NUKE.BAT.

Figure A

```
@echo off
rem NUKE.BAT completely deletes files by replacing
rem them with zero-byte files of the same name

if exist %1 goto KILL
goto OOPS

:KILL
rem > nul.dat
for %%a in (%1) do copy nul.dat %%a > nul
del nul.dat
echo File(s) permanently deleted!
goto END

:OOPS
cls
echo Check syntax. Unable to NUKE %1
echo.
echo   Enter an existing file specification,
echo   without spaces, and include the complete
echo   path to the files you want to delete
echo.
echo Examples:
echo   c:)       NUKE temp\*.txt
echo   c:\TEMP)  NUKE \wp51\altb.mac
echo   c:\WP51)  NUKE b:*.*

:END
```

The NUKE batch file completely deletes the file or files you specify.

As with most batch files, NUKE.BAT begins with the @ECHO OFF command, which keeps DOS from displaying each command in the batch file as it runs. The REM statements explain the batch file's function.

The next section determines which of two routines to execute. Essentially, it makes sure you enter an existing file specification. The line

```
if exist %1 goto KILL
```

determines whether you entered an existing file specification. If you entered a valid file specification, the batch file jumps to the :KILL routine. If you entered an invalid file specification, the command

```
goto OOPS
```

will send control to the :OOPS routine. The :OOPS routine reminds you to enter a valid file specification and shows examples of what one looks like.

The :KILL routine is the heart of NUKE.BAT. It begins with the line

```
rem > nul.dat
```

This command creates a zero-byte file. Generally speaking, DOS ignores a line that begins with REM, so it's a handy way to keep DOS from interpreting that line as a batch file command. Since DOS ignores anything you type after the REM command, you can use the redirection operator to place the line's contents—in essence, *nothing*—into a file.

NUKE.BAT creates a file named NUL.DAT. You may use any name you choose, but be very careful to use a filename that doesn't (and won't) exist on your computer. If you have a file with the same name as the zero-byte file that NUKE.BAT creates, DOS will replace your file with the zero-byte file when you run NUKE.BAT.

After NUKE.BAT creates the zero-byte file, the FOR loop

```
for %%a in (%1) do copy nul.dat %%a > nul
```

copies the zero-byte file to each file in the set you specify. The first part,

```
for %%a in (%1) do
```

sets up the loop. The parameter %1 represents the set of files you pass to the NUKE command. The variable %%a holds the current file specification during each pass through the loop. The command DO introduces the action the loop will execute. In this case, that's the command

```
copy nul.dat %%a
```

Each time through the loop, the contents of NUL.DAT will overwrite the contents of the file currently specified by the %%a variable.

Now comes the interesting part: When you copy a zero-byte file to another file, DOS deletes the target file. That's right—DOS copies NUL.DAT to your file and then deletes your original file. So, we don't need to include a separate command to delete all the files represented by the %1 parameter. We add the > nul command at the end of the FOR loop to suppress the *# file(s) copied* message.

When the loop runs out of files in %1 to process, the batch file goes to the next line,

```
del nul.dat
```

which simply deletes the empty file it created earlier. Then the batch file echoes the message *File(s) permanently deleted!* to the screen to let you know it successfully removed the file(s). The :KILL routine ends by branching to the :END routine, which ends NUKE.BAT.

As we mentioned, if you supply NUKE.BAT a nonexistent or incorrect file specification, the :OOPS routine takes over. It clears the screen and echoes the message

```
Check syntax. Unable to NUKE %1
```

where %1 is the file specification. The message also shows examples of how you should enter the command. Once :OOPS finishes, control passes to :END and the batch file quits.

Using NUKE.BAT

After you've created NUKE.BAT, it's easy to put it to work. You simply type

```
C:\>nuke file specification
```

and press [Enter]. The *file specification* should include a path if you aren't deleting files in the current directory, and it shouldn't include any spaces.

You can also use wildcards with NUKE.BAT. For example, if you want to destroy all the files in the C:\WP51\DOCS directory and you're in the C:\TEMP directory, you enter the command

```
C:\TEMP\>nuke \wp51\docs\*.*
```

If the batch file finds any files, it will delete them and display the message *File(s) permanently deleted!*

Before we can use NUKE.BAT, we need to create some sample files. Since we don't want to accidentally delete any important files, we'll place the sample files on a disk in the B: drive. Begin by placing a formatted diskette into the B: drive.

An easy way to create a sample file is to use the redirection operator (>) to echo text into a filename. To

do this, type the following commands at the C:> prompt, pressing [Enter] after each command:

```
echo I never saw a purple cow > b:annihil8.fil
echo I never hope to see one > b:obliter8.fil
echo But I can tell you anyhow > b:decim8.fil
echo I'd rather see than be one > b:gon4good.txt
```

Now that we have files, let's use NUKE.BAT to delete them. Let's nuke all the files in the root directory on the B: disk that have the FIL extension. Enter

```
C:\>nuke b:*.fil
```

After a few seconds, DOS displays the message *File(s) permanently deleted!*

Let's make a preliminary check of the batch file by switching to drive B: and running a DIR command. You should see only the file GON4GOOD.TXT in the directory list. We don't need this file either, so let's nuke it as well.

Since you're already at the B:\> prompt, you don't need to include a complete path. Just type

```
B:\>nuke gon4good.txt
```

and press [Enter].

Are they really gone?

Now that you've deleted the files, you can run the UNDELETE utility to see that there's nothing in them to recover. At the B:\> prompt, enter the command

```
B:\>undelete /dos
```

(Depending on whether you've set up deletion tracking, you may have to enter just the command or the command followed by the /DOS, /DS, or /DT switch.)

When you issue the UNDELETE command, DOS tells you how many deleted files it found and how many can be recovered, and displays the names and sizes of the deleted files. Since you copied a zero-byte file over your files, you can expect to see 0 as the size of those files. When you see the *Undelete (Y/N)?* prompt, press *n* several times to view the size of the four files you just nuked. As you can see in Figure B, each of the files you deleted contains 0 bytes of data, indicating that NUKE.BAT did its job—the files can't be recovered.

When you goof up

Although NUKE.BAT requires a valid file specification when you execute it, the batch file responds well to mistakes. To demonstrate NUKE.BAT's forgiving nature, let's run it without specifying any files. To do so, type

```
C:\>nuke
```

and press [Enter]. You'll see the message shown in Figure C.

Next, we'll supply a filename that doesn't exist. Enter the command

```
C:\>nuke b:gon4ever.txt
```

This time, you'll see the message

```
Check syntax. Unable to NUKE b:gon4ever.txt
```

Remember, when you enter an invalid file specification, NUKE automatically executes the :OOPS routine.

Figure B

```
B:\>undelete

Directory: B:\
File Specifications: *.*

    Deletion-tracking file not found.

    MS-DOS directory contains    7 deleted files.
    Of those,    3 files may be recovered.

Using the MS-DOS directory.

      ?Y_WILL          3035 11-11-93  4:12a  ...A  Undelete (Y/N)?n

      ?AR4SALE DOC     1870 11-11-93  2:01a  ...A  Undelete (Y/N)?n

      ?ESIGN   LTR      329  5-28-93  5:17p  ...A  Undelete (Y/N)?n

      ?NNIHIL8 FIL        0  2-01-80 12:00a  ...A  Undelete (Y/N)?n

      ?BLITER8 FIL        0  2-01-80 12:00a  ...A  Undelete (Y/N)?n

      ?ECIM8   FIL        0  2-01-80 12:00a  ...A  Undelete (Y/N)?n

      ?ON4GOOD TXT        0  2-01-80 12:00a  ...A  Undelete (Y/N)?
```

The UNDELETE utility shows that your deleted files are unrecoverable.

Figure C

```
Check syntax. Unable to NUKE

 Enter an existing file specification,
 without spaces, and include the complete
 path to the files you want to delete

Examples:
    c:)        NUKE temp\*.txt
    c:\TEMP)   NUKE \wp51\altb.mac
    c:\WP51)   NUKE b:*.*

C:\>
```

The :OOPS error message shows you how to specify files with NUKE.BAT.

SECOND CLASS MAIL

2020C:1144914 I:2498222 9411
FRANK RALLY
1796 GRACE AVE
SAN JOSE CA 951255620

Microsoft Technical Support
(206) 454-2030

Making TODAY.BAT restore your original directory

I created the TODAY.BAT file from the April 1993 article "Capturing the Date and Time as Environment Variables." I also implemented the correction you published in the June article "Extra TODAY.BAT Files Report the Wrong Time." Although the correction makes TODAY.BAT report the correct time, it creates the inconvenience of returning to the C:\BATCH directory instead of the directory I started from. What can I do to make TODAY.BAT restore my original directory?

John A. Dobelman
Fort Worth, Texas

In the June article Mr. Dobelman refers to, we suggested adding the command CD \BATCH to TODAY.BAT—a batch file that stores the current date and time in environment variables for use in other programs. Actually, TODAY.BAT captures the current date and time by creating a copy of itself to update the information in its directory listing. The batch file stores this copy in the current directory. The CD\BATCH command ensures that the copy replaces the TODAY.BAT file in the C:\BATCH directory. Unfortunately, this command also causes the batch file to end in the C:\BATCH directory.

However, by adding to TODAY.BAT the commands shown in red in Figure A, you can make the batch file restore your original directory. The line

```
echo @prompt cd $p$_$n: > c:\setdirec.bat
```

creates SETDIREC.BAT, which contains one command:

```
@prompt cd $p$_$n:
```

This command changes the system prompt to

```
CD \curdir
X:
```

where *curdir* is the current directory and *X* is the current drive. In doing so, it records the current directory and drive.

The next line,

```
command /c c:\setdirec.bat > c:\retdirec.bat
```

executes a second copy of COMMAND.COM, which runs the SETDIREC batch file, feeding its output,

```
CD \curdir
X:
```

to another new batch file, RETDIREC.BAT. (By using a secondary copy of COMMAND.COM to run SETDIREC.BAT, TODAY.BAT ensures that the normal system prompt remains unchanged.)

TODAY.BAT then proceeds as usual, switching to the C:\BATCH directory and recording the date and time in environment variables. When execution reaches the line

```
call c:\retdirec.bat
```

the RETDIREC batch file runs, switching back to the current directory. The last two lines,

```
del c:\setdirec.bat
del c:\retdirec.bat
```

delete the SETDIREC and RETDIREC batch files.

You can use the lines we added to TODAY.BAT in any batch file in which you want to restore your current directory. When you do, make sure the line that changes the current directory comes after the first two lines we added. The last three lines should appear when you're ready to restore the original directory.

Figure A

```
@echo off
rem TODAY.BAT creates DATE and TIME environment
rem variables.
if not "%4"=="" goto :SETDATE
echo @prompt cd $p$_$n: > c:\setdirec.bat
command /c c:\setdirec.bat > c:\retdirec.bat
cd \batch
copy /b c:\batch\today.bat+ > nul
dir c:\batch\today.bat ¦ find "TODAY" > tempdate.bat
tempdate
:SETDATE
set date=%3
set time=%4
del tempdate.bat
call c:\retdirec.bat
del c:\setdirec.bat
del c:\retdirec.bat
```

Add the red lines to make TODAY.BAT restore the original directory.

Printed in the USA
This journal is printed on recycled and recyclable paper.
50% recycled fiber including 10% post-consumer waste paper